A Tight Relationship between Generic Oracles and Type-2 Complexity Theory

Stephen Cook*

Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G4
E-mail: sacook@cs.toronto.edu

Russell Impagliazzo†

Department of Computer Science, University of San Diego, La Jolla, CA 92093
E-mail: russell@cs.ucsd.edu

and

Tomoyuki Yamakami*

Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G4
E-mail: yamakami@cs.toronto.edu

*Research supported by an NSERC operating grant and the Information Technology Research Centre.

†Research supported by NSF YI Award CCR-92-570979, Sloan Research Fellowship BR-3311, grant #93025 of the joint US-Czechoslovak Science and Technology Program, and USA-Israel BSF grant 92-00043.

Running Head: Generic Oracles and Type-2 Complexity

Proofs should be sent to: Professor Stephen Cook
Department of Computer Science
University of Toronto
Toronto, Ontario
Canada M5S 3G4

Abstract

We show that any two complexity classes satisfying some general conditions are distinct relative to a generic oracle iff the corresponding type-2 classes are distinct.

1 Introduction

Our aim in this paper is to connect type-2 complexity theory with complexity relative to a generic oracle. We begin with a general description of type-2 complexity theory.

Type-0 objects are numbers or strings, type-1 objects are functions on type-0 objects, and type-2 objects are functions on type-1 and type-0 objects. (Type-1 and type-2 objects can also be sets or relations, which we treat as a special case of functions.)

Type-2 objects occur naturally in computer science. A classic example is quadrature, which takes a real function f and numbers a and b as arguments, and produces $\int_a^b f$ as a value. The computational point of view is that f is presented as a "black box", which can produce a value $f(c)$ given a query c, but no complete description of f is provided.

The black box paradigm is appropriate whenever a complete description of the input function is large compared to the time alloted for the computation. For example, multivariate polynomials have a number of coefficients exponential in their degree. Kaltofen and Trager [16] give efficient algorithms for such things as computing greatest common divisors, when the input polynomials are accessed only through queries. Another example is NP search problems, where the input is an exponentially large search space. Beame *et al* [2] give a natural type-2 description of such problems. For a third example, Cook and Urquhart [7] show how to use higher-type polynomial-time functions to give constructive meaning to number theory theorems proved in a certain formal system.

An oracle Turing machine (OTM) is a Turing machine that is able to make queries at unit cost to its "oracle", representing a set or function. Such machines have long been used in complexity theory to represent reductions (as in Cook [6]) or relativized complexity classes (as in Baker, Gill, and Solovay [1]). The alternative point of view is to use the OTM to define a type-2 relation (or function), in which the oracle is one of the arguments of the relation. For the case of polynomial time, where the oracle represents a function whose growth affects the input size, this point of view was taken by Constable [5] and Mehlhorn [20], (see also [18] and [17]). For the case of the polynomial hierarchy, where the oracle represents a function whose growth does not affect the input size, this point of view was taken by Townsend [27], (see also [28] and [29]). For the case of **NP** search classes, this was done by Beame *et al* [2].

Baker, Gill, and Solovay [1] defined the polynomial- time hierarchy $\mathbf{PH}^A$ relativized to an oracle A. By taking our second point of view, their work defines the type-2 polynomial time hierarchy $\mathcal{PH}$, in which each member relation takes an oracle A as an argument, in addition to a string argument. Yao [30], using results from Sipser [25] and Furst, Saxe, and Sipser [11], constructed an oracle A in which all levels in $\mathbf{PH}^A$ are distinct. It follows that all levels in $\mathcal{PH}$ are absolutely distinct.

Generic sets were introduced by Cohen [4] as a tool for proving independence results in set theory. A general treatment of complexity theory relative to a generic oracle was developed by Blum and Impagliazzo [3], and Fenner *et al* [8] contains a recent survey of the subject.

In general if two complexity classes coincide relative to a generic oracle, then they coincide absolutely. This follows from a result in [3], reproduced as Theorem 2.2 below. The converse does not always hold, since for example **IP** = **PSPACE** [19], [24], but the classes are distinct relative to a generic oracle [9]. However, the question of whether a generic oracle separates two classes is natural and robust . In general, one generic oracle separates the classes iff all generic oracles separate them [3].

We prove in Theorem 3.2 below that if two type-2 classes are closed under polynomial-time many-one reductions, then they are distinct iff they are distinct when the set arguments are fixed to be some generic oracle. The special case in which the classes are members of the type-2 polynomial-time hierarchy was proved by Poizat [22]. It follows from our earlier remark that the polynomial-time hierarchy does not collapse relative to a generic oracle, a fact pointed out in [3].

In section 2 we provide basic definitions concerning type-2 classes and generic oracles, and prove an important lemma about generic oracles which is needed for our later results.

In section 3, we prove the main separation theorem mentioned above.

In section 4 we discuss relativized classes. For so-called "regular" classes, such as members of **PH**, the relativized version comes directly from the type-2 version by plugging in a fixed oracle set for the type-1 argument, so our main theorem applies directly. For "irregular" classes, such as **NP** $\cap$ **coNP** and **BPP**, the relativized version comes only indirectly from the type-2 version. However, our main theorem can still be made to apply in the important cases.

In section 5, we apply our separation result to the Townsend classes mentioned earlier. (A type-2 relation R in a Townsend class takes a type-1 function, as opposed to a relation, as an argument, but the time $T(n)$ alloted for $R's$ computation depends only on n, the length of its type-0 arguments.) In section 6 we apply our separation result to **NP** search classes.

2 Preliminaries

We consider strings over {0,1}. An *oracle* is a total function from the set of strings to the set {0,1}. A string x is said to be *in* oracle A if $A(x) = 1$. A *finite oracle* is a partial function from strings to {0,1} whose domain is finite. If σ and τ are finite oracles, we say the two are *consistent* if they agree as functions on the intersection of their two domains. If, furthermore, the domain of σ is a subset of the domain of τ, we say that τ *extends* σ (written $\tau \supseteq \sigma$). Similarly, if A is an oracle and σ is a finite oracle, A *extends* σ ($A \supseteq \sigma$) or σ is a finite prefix of A) if A and σ agree as functions on the domain of σ.

We assume that strings are ordered $s_1 < s_2 < ...$ in the usual way: first by length and then lexicographically. We restrict attention to finite oracles whose domains are initial segments in this ordering. There is a one-one correspondence between such finite oracles σ and strings x: The size $|\sigma|$ of the domain of σ equals the length $|x|$ of the string x, and for $1 \leq i \leq |\sigma|$, $\sigma(s_i)$ is the i-th bit of x. Thus we sometimes refer to a finite oracle as a string and vice versa.

A set D of finite oracles is *dense*, if every finite oracle σ has an extension to a finite oracle τ in D. The set D is *arithmetical* if there is a computable relation R on strings such that $D = \{x | Q_1 y_1 ... Q_k y_k R(x, y_1, ..., y_k)\}$ where each Q_i is a quantifier $\forall$ or $\exists$.

An oracle G is *generic* if every dense arithmetical set of finite oracles has a member σ such that $\sigma \subseteq G$.

Since there are only countably many arithmetical sets, it is a simple exercise to show that generic oracles exist. Furthermore, generic oracles are all alike from the point of view of separating complexity classes: if two classes are distinct relative to some generic oracle G, then they are distinct relative to any generic oracle. This is made precise in Theorem 1.8 of [3]. Finally, allowing access to a generic oracle does not reduce the time or space required to recognize a recursive set (see [3] and Theorem 2.2 below).

A k-ary *type-2* relation R assigns to each k-tuple $\vec{x}$ of strings and oracle X a value $R(\vec{x}, X)$ in $\{0,1\}$, where we identify 1 with "true" and 0 with "false". The relation is an *oracle property* if $k = 0$, and *type-1* if the argument X is missing. A type-2 relation R is *computable* if there is a deterministic oracle Turing machine M which, for all inputs $(\vec{x}, X)$, when $\vec{x}$ is written initially on its input tape and its query tape has access to the the oracle X, M correctly computes $R(\vec{x}, X)$. We say that R is *polynomial-time computable* if some such M computes R in time bounded by a polynomial in the length of its string inputs, where each oracle query counts as only one step in the computation. R is a Π^0_1 relation if there is a computable S such that $R(\vec{x}, X) \Longleftrightarrow \forall y S(\vec{x}, y, X)$ for all $(\vec{x}, X)$.

We say that a finite oracle σ *forces* an oracle property R if $R(A)$ holds for every oracle A extending σ. The following lemma is well-known and is the main property we use about generic oracles.

Lemma 2.1 *: Suppose R is a Π^0_1 oracle property and G is a generic oracle. Then $R(G)$ holds iff some finite prefix of G forces R.*

Proof. The $\Longleftarrow$ direction is immediate. To prove the $\Longrightarrow$ direction, suppose that $R(A) = \forall y S(y, A)$, where S is computed by a Turing machine M. Define the relation Q by

$$Q(\sigma) \Longleftrightarrow \forall y [M(y, \sigma) \downarrow \longrightarrow M(y, \sigma) = 1],$$

where $M(y, \sigma) \downarrow$ means that the computation of M on input (y, σ) is complete in the sense that all of its oracle queries are in the domain of σ. It is not hard to see that

$$\sigma \text{ forces } R \Longleftrightarrow (\forall \tau \supseteq \sigma) Q(\tau).$$

Let $D = \{\sigma | (\sigma \text{ forces } R) \vee \neg Q(\sigma)\}$. Then D is dense and arithmetical, so there exists $\sigma \in D$ such that $\sigma \subseteq G$. By assumption $R(G)$ holds, so $Q(\sigma)$ holds, and since $\sigma \in D$, it follows that σ forces R. $\Box$

As an application of this lemma, we prove a slight strengthening of Theorem 1.5 from [3].

Theorem 2.2 *Let f be any recursive function, L any recursive language, and G a generic oracle. If $L(x)$ can be computed by an oracle Turing machine with oracle G in time $f(|x|)$, then $L(x)$ is computable without an oracle in time $f(|x|)$. Similar results hold for space, nondeterministic time, and random time (either bounded or unbounded error).*

Proof. Suppose that machine M recognizes L with oracle G in time f. Let $R(X)$ assert that for all x, M with input x and oracle X halts within $f(|x|)$ steps, and accepts x iff $x \in L$. Then R is a Π_1^0 oracle property such that $R(G)$ holds. Hence some finite prefix σ of G forces R. Let M' be the modification of M which answers each oracle query in the domain of σ according to σ, and answers "no" to every other oracle query. Then M' is a Turing machine without oracle which recognizes L. If M' is carefully constructed, it can keep track within its finite state control (using no extra steps) what is written on its query tape, provided that that string is an initial segment of something in the domain of σ. Thus M' is ready to answer each query immediately, and so M' recognizes L within time f, as required. $\square$

3 The Basic Separation Result

For each type-2 relation R and oracle A we define the type-1 relation $R[A]$ by $R[A](\vec{x}) = R(\vec{x}, A)$. If $\mathcal{C}$ is a class of type-2 relations and A is an oracle, then $\mathcal{C}[A] = \{R[A] | R \in \mathcal{C}\}$. Obviously if $\mathcal{C}$ and $\mathcal{D}$ are type-2 classes such that $\mathcal{C} \subseteq \mathcal{D}$, then $\mathcal{C}[A] \subseteq \mathcal{D}[A]$ for any oracle A. We wish to find conditions on classes so that the converse holds when A is generic.

Definition 3.1 (polynomial-time many-one reduction) Let R and S be type-2 relations, and suppose that R is k-ary and S is j-ary. Then R is *polynomial-time many-one reducible to* S, denoted $R \leq_m^p S$, if there exist type-2 polynomial-time computable functions $F_1, ..., F_j$ and a type-2 polynomial-time computable relation Q such that

$$R(\vec{x}, A) = S(\vec{F}(\vec{x}, A), Q[\vec{x}, A])$$

where $\vec{F}(\vec{x}, A) = (F_1(\vec{x}, A), ..., F_j(\vec{x}, A))$ and $Q[\vec{x}, A] = \lambda z.Q(\vec{x}, z, A)$.

We say that a class $\mathcal{C}$ of type-2 relations is *closed under* $\leq_m^p$ if for all type-2 relations R and S, if $R \leq_m^p S$ and $S \in \mathcal{C}$ then $R \in \mathcal{C}$.

If follows from the next result that the polynomial hierarchy relative to a generic oracle does not collapse, since it is known that the type-2 polynomial hierarchy does not collapse (see Section 1).

Theorem 3.2 *Let $\mathcal{C}$ and $\mathcal{D}$ be classes of computable type-2 relations and suppose that $\mathcal{C}$ and $\mathcal{D}$ are closed under $\leq_m^p$. Then for any generic oracle G,*

$$\mathcal{C} \subseteq \mathcal{D} \Longleftrightarrow \mathcal{C}[G] \subseteq \mathcal{D}[G].$$

Proof. The direction $\Longrightarrow$ is immediate and does not depend on G being generic or the classes being closed. To prove the direction $\Longleftarrow$, we make two definitions: For an oracle A and a finite oracle σ, let A^σ be the same as A except on the domain of σ, A^σ coincides with σ. Given a k-ary relation R we define a $(k+1)$-ary relation $\hat{R}$ by $\hat{R}(\vec{x}, \sigma, X) = R(\vec{x}, X^\sigma)$. Notice that $\hat{R} \leq_m^p R$.

Assume that $\mathcal{C}[G] \subseteq \mathcal{D}[G]$ and $R \in \mathcal{C}$. Then $\hat{R} \in \mathcal{C}$ and hence $\hat{R}[G] \in \mathcal{C}[G]$ so $\hat{R}[G] \in \mathcal{D}[G]$. Hence there is $S \in \mathcal{D}$ such that $R(\vec{x}, G^\sigma) = S(\vec{x}, \sigma, G)$ for all $\vec{x}, \sigma$. Now let

$$T(X) \equiv \forall \vec{x} \forall \sigma [R(\vec{x}, X^\sigma) = S(\vec{x}, \sigma, X)].$$

Then T is a Π_1^0 relation, and hence by Lemma 2.1 there is $\tau \subseteq G$ such that $T(A)$ holds for all $A \supseteq \tau$. Now define $\sigma(X)$ to be the unique σ such that $|\sigma| = |\tau|$ and $\sigma \subseteq X$. Note that $\sigma(X)$ is a polynomial-time computable function of X. But then for all $\vec{x}$ and X, $R(\vec{x}, X) = S(\vec{x}, \sigma(X), X^\tau)$, so $R \leq_m^p S$. Therefore $R \in \mathcal{D}$. □

4 Relativized Classes

A typical type-1 complexity class $\mathbf{C}$ has a natural type-2 counterpart $\mathcal{C}$, based on the same resource bounds used to define $\mathbf{C}$, and, for each oracle A, a natural relativized version $\mathbf{C}^A$. Let us say that $\mathbf{C}$ is *regular* if

$$\mathbf{C}^A = \mathcal{C}[A] \tag{1}$$

for all A. For example, the classes in the polynomial time hierarchy are regular. In particular, the type-2 counterpart of $\mathbf{P}$ is the class $\mathcal{P}$ of relations $R(\vec{x}, X)$ computable by a deterministic polynomial-time Turing machine with access to the oracle X, and similarly the type-2 counterpart of $\mathbf{NP}$ is $\mathcal{NP}$, where now we allow the oracle Turing machine to be nondeterministic. For each oracle A, the relativized type-1 class $\mathbf{P}^A$ is $\mathcal{P}[A]$ and $\mathbf{NP}^A$ is $\mathcal{NP}[A]$.

If $\mathbf{C}$ and $\mathbf{D}$ are regular classes (or at least classes for which (1) holds when A is generic) with suitable closure properties, then Theorem 3.2 implies that for generic G,

$$\mathcal{C} \subseteq \mathcal{D} \Longleftrightarrow \mathbf{C}^G \subseteq \mathbf{D}^G. \tag{2}$$

Examples of irregular class are $\mathbf{NP} \cap \mathbf{coNP}$ and $\mathbf{BPP}$. The natural relativized version of $\mathbf{NP} \cap \mathbf{coNP}$ is $\mathbf{NP}^A \cap \mathbf{coNP}^A$ and the natural type-2 version is $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$, but below we construct an oracle A and a relation R such that $R[A]$ is in $\mathbf{NP}^A \cap \mathbf{coNP}^A$ but $R[A]$ is not in $(\mathcal{NP} \cap \mathrm{co}\mathcal{NP})[A]$.

A language in $\mathbf{BPP}$ is defined by a probabilistic Turing machine for which the probabilities of acceptance and rejection are bounded away from one half. For the type-2 counterpart $\mathcal{BPP}$ we require that these probabilities be uniformly bounded away from a half for all oracles A. But for each oracle A, to show membership in the relativized class $\mathbf{BPP}^A$ we only require that the probabilities be bounded away for that particular A.

Proposition 4.1 *The classes* **NP**$\cap$**coNP** *and* **BPP** *are irregular. Thus there exist oracles A and B such that* $\mathbf{NP}^A \cap \mathbf{coNP}^A \neq (\mathcal{NP} \cap \mathrm{co}\mathcal{NP})[A]$ *and* $\mathbf{BPP}^B \neq \mathcal{BPP}[B]$.

Proof. To show that **NP**$\cap$**coNP** is irregular, we use the fact ([3],[14],[26], [29]) that $(\mathcal{NP}\cap\mathrm{co}\mathcal{NP})[A] \subseteq \mathbf{P}^{\mathbf{NP}\oplus A}$ for any oracle set A, where $A \oplus B =_{df} \{0x \mid x \in A\} \cup \{1x \mid x \in B\}$. Thus it suffices to construct A so that $\mathbf{NP}^A \cap \mathbf{coNP}^A \not\subseteq \mathbf{P}^{SAT\oplus A}$, where SAT is any **NP**-complete problem.

Baker, Gill, and Solovay [1] construct an oracle A separating $\mathbf{P}^A$ and $\mathbf{NP}^A \cap \mathbf{coNP}^A$. This construction is easily relativized so that we can make the separating oracle A have the form $SAT \oplus B$ for some B. Thus $\mathbf{NP}^A \cap \mathbf{coNP}^A \not\subseteq \mathbf{P}^A$, and since $\mathbf{P}^A = \mathbf{P}^{SAT\oplus A}$, A meets our requirements.

The argument that **BPP** is irregular is similar, but we use the fact [15] that $\mathcal{BPP}[B] \subseteq \mathbf{P}^{\mathbf{PSPACE}\oplus B}$ for any oracle B. We then relativize the construction [23] of an oracle B separating **BPP** from **P** to apply to an oracle B of the form $QBF \oplus C$, where QBF is any **PSPACE**-complete problem (such as the quantified Boolean formula problem). $\square$

Even though $\mathbf{NP} \cap \mathrm{co}\mathbf{NP}$ and **BPP** are irregular, we now show that (1) holds for the important case in which A is generic, and so (2) can be applied. This shows, for example, that $\mathbf{P}^G = \mathbf{NP}^G \cap \mathbf{coNP}^G$ for generic G iff $\mathcal{P} = \mathcal{NP} \cap \mathrm{co}\mathcal{NP}$, which is part of Theorem 2.2 of [15].

Proposition 4.2 *For any generic oracle G,* $\mathbf{NP}^G \cap \mathbf{coNP}^G = (\mathcal{NP} \cap \mathrm{co}\mathcal{NP})[G]$, *and* $\mathbf{BPP}^G = \mathcal{BPP}[G]$.

Proof. The right sides are clearly subsets of the left sides. We prove the reverse inclusion for the first equation. The proof for the second is similar.

Suppose $R(\vec{x})$ is in $\mathbf{NP}^G \cap \mathbf{coNP}^G = \mathcal{NP}[G] \cap \mathrm{co}\mathcal{NP}[G]$. Then there are polynomial-time type-2 relations S and T and a polynomial p such that for all $\vec{x}$, $R(\vec{x}) = ES(\vec{x}, G) = AT(\vec{x}, G)$, where $ES(\vec{x}, X) \equiv \exists y \leq p(|\vec{x}|)S(\vec{x}, y, X)$ and $AT(\vec{x}, X) \equiv \forall y \leq p(|\vec{x}|)T(\vec{x}, y, X)$.

Define the oracle property Q by

$$Q(X) \equiv \forall \vec{x}[ES(\vec{x}, X) = AT(\vec{x}, X)].$$

Then Q is a Π^0_1 relation such that $Q(G)$ holds, so by Lemma 2.1 there is a finite prefix σ of G such that $Q(A)$ holds for all $A \supseteq \sigma$.

Recall the definition of X^σ from the proof of Theorem 3.2, and note that $X^\sigma \supseteq \sigma$, so $Q(X^\sigma)$ holds for all X. Define R_1 by

$$R_1(\vec{x}, X) \equiv ES(\vec{x}, X^\sigma) \equiv AT(\vec{x}, X^\sigma).$$

Then R_1 is in $\mathcal{NP} \cap \mathrm{co}\mathcal{NP}$, and since $G^\sigma = G$ we conclude that $R = R_1[G]$, as required. $\square$

One can also prove versions of Proposition 4.2 for the classes **RP**, $\mathbf{BPP}^{\mathbf{NP}}$ etc. Although we do not have a complete characterization of which classes will have this property, the following generalizes the above argument somewhat:

Proposition 4.3 *Let C and C' be classes of computable type-2 relations closed under many-one reductions. Then for any generic oracle G, $C[G] \cap C'[G] = (C \cap C')[G]$.*

In particular, if C, C' are regular classes, $\mathcal{C}^G = C[G]$ and $\mathcal{C}'^G = C'[G]$ so this becomes: $\mathcal{C}^G \cap \mathcal{C}'^G = (C \cap C')[G]$.

Proof. The proof is basically that above. If $R(\vec{x})$ is in $C[G] \cap C'[G]$, then there are type 2 relations $S \in C$ and $T \in C'$ so that $S(\vec{x}, G) = T(\vec{x}, G)$ for all $\vec{x}$. Since S and T are computable, by Lemma 2.1 there is a finite prefix σ of G such that $S(\vec{x}, A) = T(\vec{x}, A)$ holds for all $A \supseteq \sigma$.

Define R_1 by

$$R_1(\vec{x}, X) \equiv S(\vec{x}, X^\sigma) \equiv T(\vec{x}, X^\sigma).$$

Then R_1 is in $C \cap C'$, and $R = R_1[G] \in (C \cap C')[G]$, as required. □

5 Townsend Classes

Sometimes it is convenient to allow a function argument $\alpha : \Sigma^* \longrightarrow \Sigma^*$ in a relation $R(\vec{x}, \alpha)$ in place of a set argument X. Townsend [27] defined a type-2 version of the polynomial hierarchy based on this idea (see also [28, 29]). We follow Townsend (in contrast to Mehlhorn [20]; see also [5], [17] and [12]) in ignoring the oracle α in allotting time to a Turing machine with oracle α. That is, we say that a Turing machine M with oracle α which computes $R(\vec{x}, \alpha)$ is $T(n)$ time-bounded provided that for all $\vec{x}$ and for all α, M halts within $T(n)$ steps, where n is the length of $\vec{x}$. In particular, if M operates in polynomial time, then M can examine only a polynomial length prefix of any oracle value $\alpha(x)$ during any computation. This motivates the following definitions.

Definition 5.1 (Townsend relation) Let p be a polynomial and R be a relation with string arguments $\vec{x}$ and function argument α. We say that R *has dependency bounded by* p if for all $\vec{x}$ and α, $R(\vec{x}, \alpha) = R(\vec{x}, \alpha_{p(|\vec{x}|)})$, where $\alpha_t(y)$ is the first t symbols of $\alpha(y)$ (or $\alpha(y)$ if $t > |\alpha(y)|$). We say that R is a *Townsend relation* iff R has dependency bounded by some polynomial p.

For example, each class of the Townsend polynomial hierarchy [27] is a class of Townsend relations.

We can translate back and forth from functions to sets as follows. Assume some efficient way of encoding triples (x, ℓ, i) by strings $< x, \ell, i >$, where $x \in \{0,1\}^*$, $\ell \in \{1\}^*$, and $i \in \{1, 2\}$. We define a transformation from a function α to a set $A_\alpha \subseteq \{0,1\}^*$ by

$$A_\alpha = \{< x, \ell, 1 >: \text{bit number } |\ell| \text{ of } \alpha(x) \text{ is } 1\} \bigcup \{< x, \ell, 2 >: |\alpha(x)| = |\ell|\}.$$

Given a positive integer t we define a transformation taking a set A to a function $\beta[A, t]$ by the conditions

$$|\beta[A, t](y)| = \min_i [< x, 1^i, 2 > \in A \text{ or } i = t]$$

and for $1 \leq i \leq |\beta[A, t](y)|$ the i-th bit of $\beta[A, t](y)$ is 1 iff $< x, 1^i, 1 > \in A$.

Note that for all α, t, y,

$$\beta[A_\alpha, t](y) = \alpha_t(y) \tag{3}$$

where as above $\alpha_t(y)$ is the first t symbols of $\alpha(y)$.

We extend Definition 3.1 of $\leq_m^p$ to Townsend relations by replacing the argument A by α and replacing the relation Q by a polynomial time second-order function F. Note that the notion of polynomial time ignores the growth rate of oracle arguments α, as mentioned above.

If R is a Townsend relation and A is a set, then we interpret $R(\vec{x}, A)$ by interpreting A as the characteristic function of the set A: that is $A(x) = 1$ if $x \in A$ and $A(x) = 0$ if $x \notin A$, where 0 and 1 are strings of length one. Thus $R[A]$ and $\mathcal{C}[A]$ make sense, where R is a Townsend relation and $\mathcal{C}$ is a Townsend class (i.e. class of Townsend relations). If R is a Townsend relation, we define the corresponding type-2 relation R^{set} by $R^{\text{set}}(\vec{x}, A) = R(\vec{x}, A)$, where again the second occurrence of A refers to the characteristic function of the set A. If $\mathcal{C}$ is a Townsend class, then $\mathcal{C}^{\text{set}}$ is the class of corresponding type-2 relations.

Theorem 5.2 *Let $\mathcal{C}$ and $\mathcal{D}$ be classes of computable Townsend relations and suppose that $\mathcal{C}$ and $\mathcal{D}$ are closed under $\leq_m^p$. Then for any generic oracle G,*

$$\mathcal{C} \subseteq \mathcal{D} \Longleftrightarrow \mathcal{C}[G] \subseteq \mathcal{D}[G].$$

Proof. As before the direction $\Longrightarrow$ is immediate. To prove the direction $\Longleftarrow$, we translate the Townsend classes $\mathcal{C}$ and $\mathcal{D}$ to the corresponding type-2 classes $\mathcal{C}^{\text{set}}$ and $\mathcal{D}^{\text{set}}$ defined above, and apply Theorem 3.2. Notice that the hypotheses of the present theorem imply that $\mathcal{C}^{\text{set}}$ and $\mathcal{D}^{\text{set}}$ satisfy the hypotheses of Theorem 3.2.

Suppose $\mathcal{C}[G] \subseteq \mathcal{D}[G]$. Then $\mathcal{C}^{\text{set}}[G] \subseteq \mathcal{D}^{\text{set}}[G]$. Hence by Theorem 3.2,

$$\mathcal{C}^{\text{set}} \subseteq \mathcal{D}^{\text{set}}. \tag{4}$$

Now suppose $R \in \mathcal{C}$. By definition of *Townsend relation*, there is a polynomial p such that R has dependency bounded by p. Define the Townsend relation R_0 by $R_0(\vec{x}, A) = R(\vec{x}, \beta[A, p(|\vec{x}|)])$ when A is a characteristic function of a set, and more generally $R_0(\vec{x}, \alpha) = R_0(\vec{x}, A)$, where $A = \{x : \alpha(x) = 1\}$. Then $R_0 \leq_m^p R$, so $R_0 \in \mathcal{C}$. Hence $R_0^{\text{set}} \in \mathcal{C}^{\text{set}}$, so by (4) $R_0^{\text{set}} \in \mathcal{D}^{\text{set}}$. Since $\mathcal{D}$ is closed under $\leq_m^p$, it is easy to see that $R_0 \in \mathcal{D}$. But $R \leq_m^p R_0$, because by Definition 5.1 and (3), $R(\vec{x}, \alpha) = R(\vec{x}, \alpha_{p(|\vec{x}|)}) = R(\vec{x}, \beta[A_\alpha, p(|\vec{x}|)]) = R_0(\vec{x}, A_\alpha)$. Therefore $R \in \mathcal{D}$. $\square$

It follows from the above theorem that the Townsend polynomial hierarchy does not collapse relative to a generic oracle, because Yao's oracle [30] separates the type-2 hierarchy. Fortnow and Yamakami [10] strengthen this result to show that $\Sigma_k^p \cap \Pi_k^p$ properly contains Δ_k^p relative to a generic oracle. From the above theorem we can conclude proper containment for the corresponding type-2 classes.

6 Search Classes

For this section we follow the treatment in [2] of type-2 search problems, motivated by Papadimitriou's **NP** search classes [21]. In general, a type-2 search problem Q assigns a set $Q(x, \alpha)$ of strings to (x, α), representing the set of possible solutions to problem instance (x, α). We say that Q is an **NP** search problem if the relation $R(x, y, \alpha) \equiv y \in Q(x, \alpha)$ is polynomial-time computable, and if in addition there is a polynomial p such that each $y \in Q(x, \alpha)$ satisfies $|y| \leq p(|x|)$. Q is *total* if $Q(x, \alpha)$ is nonempty for all x and α.

We let $\mathcal{TFNP}$ denote the class of total type-2 **NP** search problems.

An example of a problem in $\mathcal{TFNP}$ is *LEAF*. An argument (x, α) for *LEAF* codes an undirected graph G with degree at most two whose nodes are strings of length $|x|$ or less, such that the node $0^{|x|}$ is a leaf, called the *standard* leaf. Then G must have at least one other leaf, and in fact $LEAF(x, \alpha)$ is the set of nonstandard leaves in G.

The problem *SINK* is defined similarly, but now G is directed with maximum indegree and outdegree one, $0^{|x|}$ is a source, and $SINK(x, \alpha)$ is the set of sinks in G.

Informally, a search problem Q_1 is reducible to a search problem Q_2 if any solution to a transformed instance of Q_2 can be transformed to a solution to Q_1.

Definition 6.1 (search reduction) Let Q_1 and Q_2 be type-2 search problems. Then Q_1 *is polynomial-time many-one reducible to* Q_2, denoted $Q_1 \leq_m^p Q_2$, if there exist type-2 polynomial-time computable functions F, G, and H, such that for all x, y, and α,

$$y \in Q_2(F(x, \alpha), G[x, \alpha]) \Longrightarrow H(x, y, \alpha) \in Q_1(x, \alpha)$$

where $G[x, \alpha] = \lambda z.G(x, z, \alpha)$.

We say that a subclass $\mathcal{C}$ of $\mathcal{TFNP}$ is *closed under* $\leq_m^p$ if for all problems Q_1 and Q_2 in $\mathcal{TFNP}$, if $Q_1 \leq_m^p Q_2$ and $Q_2 \in \mathcal{C}$ then $Q_1 \in \mathcal{C}$.

Theorem 6.2 *Let $\mathcal{C}$ and $\mathcal{D}$ be subclasses of $\mathcal{TFNP}$ which are closed under $\leq_m^p$. Then for every generic oracle G,*

$$\mathcal{C} \subseteq \mathcal{D} \Longleftrightarrow \mathcal{C}[G] \subseteq \mathcal{D}[G].$$

Proof. As before, the $\Longrightarrow$ direction is immediate. For the converse, assume that $\mathcal{C}[G] \subseteq \mathcal{D}[G]$ and $Q \in \mathcal{C}$. Define the relation R by $R(x, y, \alpha) \equiv y \in Q(x, \alpha)$. Proceed as in the proof of Theorem 3.2 to handle the case in which the function argument α is a set A, and then apply the technique of the proof of Theorem 5.2 to handle the general case. $\square$

The above result is used to prove Theorem 1 of [2] (there stated without proof). It follows from this and other results in [2] that a number of **NP** search problems are distinct relative to a generic oracle. For example, the class **PPA** of problems reducible to *LEAF* is distinct from the class **PPADS** of problems reducible to *SINK*, relative to a generic oracle.

Acknowledgments

Lance Fortnow suggested this topic to the third author, who wrote the preliminary version of this paper.

References

[1] Baker, T. J., Gill, J., and Solovay, R. (1975), Relativizations of the $P = ?NP$ question, *SIAM J. Computing* **4**, 431–442.

[2] Beame, P., Cook, S., Edmonds, J., Impagliazzo, R., and Pitassi T. (1995), The relative complexity of NP search problems, *in* "Proceedings, 27th ACM Symposium on Theory of Computing", pp. 303–314.

[3] Blum, M., and Impagliazzo, R. (1987), Generic oracles and oracle classes, *in* "Proceedings, 28th IEEE Symposium on Foundations of Computer Science", pp. 118–126.

[4] Cohen, P. (1963; 1964), The independence of the continuum hypothesis I, II, *in* "Proceedings, Nat. Acad. Sci. U.S.A." **50**, pp. 1143–1148; **51**, pp. 105–110.

[5] Constable, R. L. (1973), Type two computable functionals, *in* "Proceedings, 5th ACM Symposium on Theory of Computing", pp. 108–121.

[6] Cook, S. (1971), The complexity of theorem-proving procedures, *in* "Proceedings, 3rd ACM Symposium on Theory of Computing", pp. 151–158.

[7] Cook, S. and, Urquhart, A. (1993) Functional interpretaions of feasibly constructive arithmetic, *Annals of Pure and Applied Logic* **63**, 103–200.

[8] Fenner, S., Fortnow, L., Kurtz, S. A., and Li, L. (1993), An oracle builder's toolkit, *in* "Proceedings, 8th IEEE Conference on Structure in Complexity Theory", pp. 120–131.

[9] Fortnow, L., and Sipser, M. (1988), Are there interactive protocols for co-NP languages?, *Information Processing Letters* **28**, 249–251.

[10] Fortnow, L., and Yamakami, T. (1996) Generic separations, *J. Comput. System Sci.* **52**, 191–197.

[11] Furst, M., Saxe, J. B., and Sipser, M. (1984), Parity, circuits, and the polynomial time hierarchy, *Math. Systems Theory* **17**, 13–27.

[12] Harnik, V. (1992), Provably total functions of intuitionistic bounded arithmetic, *J. Symbolic Logic* **57**, 466–477.

[13] Hartmanis, J., and Hemachandra, L. A. (1987), One-way functions, robustness, and the non-isomorphism of NP-complete sets, *in* "Proceedings, 2nd Conference on Structure in Complexity Theory", pp. 160–174.

[14] Hartmanis, J., and Hemachandra, L. A. (1990), Robust machines accept easy sets, *Theoret. Comput. Sci.* **74**, 217–225.

[15] Impagliazzo, R., and Naor, M. (1988), Decision trees and downward closures, *in* "Proceedings, 3rd IEEE Conference on Structure in Complexity Theory", pp. 29–38.

[16] Kaltofen, E., and Trager, B. (1990) Computing with polynomials give by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators, *J. Symbolic Computation* **9**, 301–320.

[17] Kapron, B. M., and Cook, S. A. (1996), A new characterization of type-2 feasibility, *SIAM J. Computing* **25**, 117–132.

[18] Ko, K. (1985), On some natural complete operators, *Theor. Comput. Sci.* **37**, 1–30.

[19] Lund, C., Fortnow, L., Karloff, H., and Nisan, N. (1992), Algebraic methods for interactive proof systems, *J. ACM* **39**, 859–868.

[20] Mehlhorn, K. (1976), Polynomial and abstract subrecursive classes, *J. Comput. System Sci.* **12**, 147–178.

[21] Papadimitriou, C. H. (1990), On graph-theoretic lemmata and complexity classes, *in* "Proceedings, 31st Symposium on Foundations of Computer Science", pp. 794–801.

[22] Poizat, B. (1986), $\mathcal{Q} = \mathcal{NQ}$?, *J. Symbolic Logic* **51**, 22–32.

[23] Rackoff, C. (1982), Relativized questions involving probabilistic algorithms, *J. ACM* **29**, 261–268.

[24] Shamir, A. (1991), IP = PSPACE, *J. ACM* **39**, 869–877.

[25] Sipser, M. (1983), Borel sets and circuit complexity, *in* "Proceedings, 15th ACM Symposium on Theory of Computing", pp. 61–69.

[26] Tardos, G. (1989), Query complexity, or why is it difficult to separate $\mathrm{NP}^A \cap \text{co-NP}^A$ from P^A by random oracle A ?, *Combinatorica* **9**, 385–392.

[27] Townsend, M. (1990), Complexity for type-2 relations, *Notre Dame J. Formal Logic* **31**, 241–262.

[28] Yamakami, T. (1995), Feasible computability and resource bounded topology, *Information and Computation* **116**, 214–230.

[29] Yamakami, T. (1992), Structural properties for feasibly computable classes of type two, *Math. Systems Theory* **25**, 177–201.

[30] Yao, A. (1985), Separating the polynomial-time hierarchy by oracles, *in* "Proceedings, 26th IEEE Symposium on Foundations of Computer Science", pp. 1–10.